OULU UNIVERSITY OF
APPLIED SCIENCES

Mikko Gronoff

# OPEN SOURCE MAZEBALL GAME FOR WINDOWS PHONE

# OPEN SOURCE MAZEBALL GAME FOR WINDOWS PHONE

Mikko Gronoff
Master's Thesis
Spring 2013
Degree Programme In Information Technology
Oulu University of Applies Sciences

# ABSTRACT

Oulu University of Applied Sciences
Degree Programme In Information Technology

---

Author: Mikko Gronoff
Title of Master's thesis: Open Source Mazeball Game for Windows Phone
Supervisor: Kari Laitinen
Term and year of completion: Spring 2013 Number of pages: 45 + 2 appendices

---

Mobile gaming has developed in huge leaps in the last 10 years. More powerful handheld gaming devices and mobile phones have brought an advanced audiovisual gaming content to these devices which are shipping in hundreds of millions a year. Mobile gaming market is easy to enter as development tools are free or inexpensive, a testing unit can usually be found in a developer's own pocket, and device vendors are openly accepting more variety to their online stores.

The purpose of this Master´s thesis was to present a full cycle of an open source mobile platform game development project from a rough base idea to the development process and finally to the completion of the game. The thesis was divided into four main chapters related to the game development project: Windows Phone as a game development platform, an open source, a game design and a game development. The research question of this research was "How can open source be utilized in a Windows Phone game development?".

The research was made by developing a mazeball game for the Windows Phone platform utilizing as much open source resources as possible. As a result, it became clear that open source can be utilized in multiple ways in a game development project. One way is to use available licensed resources. This means that there are various websites offering different kind of audio and graphical content for varying degree of freedom. Another way is the open source software itself that allows a developer to study, use and derive an existing code. A third way is to use open source tools. These tools may offer equal or even better features and functions than commercial closed source counterparts.

As a conclusion it was surprising to find such a variety of open source software and licensed resources available for game development, and to see there are open source based tools for most needs a student or developer can imagine from a 3D modeling to a full office suite. Further development possibilities for the Mazeball game are vast, starting with finalizing the maze level 3D model to contain textures for walls and floor, then further expanding the game by adding new levels and functions.

---

Keywords: game design, game development, open source, programming, Windows Phone

# CONTENTS

**TERMS AND ABBREVIATIONS**

| | |
|---|---|
| API | Application Programming Interface. An interface consisting of a set of routines, protocols and tools that software components can use to communicate with each other. |
| Apache 2.0 | Popular open source software license. |
| BSD-New | Popular open source software license. |
| GPL | General Public License. The most popular open source software license. |
| FLOSS | Free/Libre/Open Source Software. Software that is both free and open source. |
| FSF | Free Software Foundation. A foundation promoting open source software distribution under GPL license. |
| IDE | Integrated Development Environment. Software application providing tools for developers for software development. Usually consists of a source code editor, a builder and a debugger. |
| iOS | A mobile operating system by Apple Inc. |
| LGPL | Lesser General Public License. Popular open source software license. Less restrictive than GPL, allowing a linked usage with an open source software component licensed under a different license than GPL. |
| MIT/X11 | Popular open source software license. |
| OS | Operating System. |
| OSS | Open Source Software. |
| SDK | Software Development Kit. A set of software development tools, usually including IDE and other tools. |
| UI | User Interface. A system by which user interacts with a machine. Provides means for input and output, such as a keyboard or a screen. |
| XNA | XNA Framework, a set of managed libraries designed for the game development based on the Microsoft .NET Framework. |

# 1 INTRODUCTION

Mobile gaming has developed in huge leaps in the last 10 years. More powerful handheld gaming devices and mobile phones have brought an advanced audiovisual gaming content to these devices which are shipping in hundreds of millions a year. The latest addition to this device "family", a tablet, has narrowed the border between handheld devices and more traditional gaming devices, such as personal computers and game consoles. Mobile gaming market is easy to enter as development tools are free or inexpensive, a debug and testing unit can usually be found in developer's own pocket and device vendors are openly accepting more variety to their online stores.

I have been an enthusiastic gamer for all my life. Board games, computer games, console games and role playing games among others have always been an important hobby for me. The idea for doing a digital representation of a real-life childhood favourite, a mechanical maze game, had been with me for a few years. I was supposed to do it with a friend for another mobile platform related to other studies but due time constraints the idea was shelved. Being a relatively inexperienced software developer, the old idea about the game sounded right: a personally motivating idea for enhancing my programming skills, the fact that game development is an evolving, expanding business in Finland (Neogames 2011, 6 - 10) and a type of game that can be done iteratively, adding new features and all kinds of bells and whistles so to speak if the set time schedule allows after the main functionality has been done.

Windows Phone 7 from Microsoft was chosen as a development platform for its strong support of the game development by XNA framework and for the fact that I had the device available. The XNA framework, also supported by the XBOX360 game console, seemed to be easy to get into and powerful in a 3D game development.

Open source for me personally is something I have been interested in for years and finally after doing a brief open source team work during Master's Degree studies, I got the spark for doing this thesis work as open source, by utilizing open source and related licensed resources. The research question of this thesis is **How can open source be utilized in a Windows Phone game development?**.

The terms "open source", "free software" and "free/libre open source" are often used when talking

about the same thing even though open source does not mean necessarily free and vice versa. In this thesis the term "open source" is used as a wider umbrella term including in addition to software also resources licensed in a similar manner like music, graphics and literary works. All of these resources, when published using open source or similar licenses, share the same principal idea of openness, free usage and common good.

The main purpose of this thesis was to present the reader a full cycle of application development from the rough base idea to the development process and finally to the completion of the game. The thesis is divided into four main chapters related to the game development project: Windows Phone as a game development platform, open source, game design and game development.

Chapter 2 covers the technical overview of Windows Phone platform and XNA framework, especially how the latter supports game development and what can be done with it. Topics include platform features, supported media formats and 3D graphics among others.

Chapter 3 deals with open source in general. A chapter focused on this is a necessity in this thesis, as the game development project documented in chapter 4 is purely open source and utilizes open source resources and tools. The chapter covers the very important topic of licensing and also offers a few real-life examples of possible legal issues.

Chapter 4 presents an overview of game design fundamentals. The rules of game design are discussed as well as different parts of a core gameplay design such as a control system and a level creation. A chart of a general game project design and creation flow is presented.

Chapter 5 describes the developed Windows Phone 7 game MazeBall starting with the initial idea and requirements that were set before beginning the development. The development process is reported and different parts of the game are looked at. The usage of different open source tools and resources in the game are covered. Code samples and figures are used to further detail some parts of the game and used resources. The answer for the research question is summarized and illustrated results of the study are given.

Finally, Chapter 6 includes thoughts on the development process and the challenges and benefits of using open source. Further development possibilities of the game project are discussed as well as shortcomings of open source tools.

This thesis is useful to students and other people interested in open source and game development in general or specifically for Windows Phone. All code of the game development project is distributed as open source under Microsoft Public License and available on http://mazeball.codeplex.com with other licensed content including music, sound effects, graphics and *.blend* files.

# 2 WINDOWS PHONE

## 2.1 Platform overview

Windows Phone is a mobile phone platform by Microsoft. The history of the platform dates back to 2002 when the first mobile phones running Windows Phone predecessor Windows Mobile operating system were released. Windows Mobile never got a huge market share among mobile phone OS's but Microsoft kept updating it and releasing new versions through the years. In 2010, Windows Phone 7 was annouced as a successor to the old Windows Mobile OS. The latest OS version, Windows Phone 8, was released in late 2012.

Current mobile device industry leaders Samsung and Apple had globally a combined smartphone market share of roughly 90% at the end of year 2012 making Windows Phone a minor player in the global mobile OS competition with 3% market share (Gartner 2013, date of retrieval 19.3.2013). Since Nokia chose Windows Phone as the new primary smartphone OS in February 2011, the platform has gained a lot more publicity, marketing and overall interest than before. Of all Windows Phone devices in the market as of March 2013, about 78% were Nokia Lumia devices (AdDuplex 2013, date of retrieval 20.3.2013). Only recently there has been some news on Windows Phone gaining stronger ground among competition at least in Europe. Italy and UK have been adopting the OS well (Kantar 2013, date of retrieval 19.3.2013) but it remains to be seen what the future will be for Windows Phone.

The XNA Game Studio makes game development for Windows Phone quite attractive. The Game Studio is a programming environment that allows creation of games for Windows Phone, Xbox 360 and Windows desktop operating system. XNA Game Studio includes the XNA Framework, a set of managed libraries designed for game development based on the Microsoft .NET Framework (MSDN 2013a, date of retrieval 20.3.2013). By utilizing the XNA framework the same code can be used for all three platforms with only minimal changes. This results in easy porting from one device to another, and in theory allows bigger market for the product and possibility to get the same game experience for mobile and desktop platforms. This is especially true with new Windows 8 and Windows Phone 8 where the user interface is very similar in both OSs, thus blurring the line between different devices. In an ideal situation this reduces the importance of a specific device used for gaming when the user gets the same experience, independent of the device. However, upon release of Windows 8 Microsoft has taken a step back in the

interoperability of XNA framework (Gamasutra 2013, date of retrieval 20.3.2013) as they propose DirectX 11 as the standard framework for game development for desktop OS, even though the overall UI design has been streamlined between OSs. Hence, the synergy of 3 platforms with one codebase is truly a fact only for Windows Phone 7/8, Windows Vista/7 and Xbox 360.

One very interesting thing related to XNA framework and open source is MonoGame, an open source implementation of XNA framework for other platforms. The goal of MonoGame project is to offer a high code reuse possibility for developers across different platforms (MonoGame 2013, date of retrieval 20.3.2013). In practise, developing games using XNA framework opens a way to create games also for iOS, Android, Mac OS X and other platforms in addition to Windows using the same code base.

### 2.2 Development requirements

Minimum software requirements needed for Windows Phone development are Windows Vista or a newer operating system and a Windows Phone 7.1 Software Development Kit, or SDK, including all the necessary tools for the phone software development like an emulator, an integrated development environment (IDE), and additional tools. Zune, a digital media software suite from Microsoft, is needed for transferring application from a computer to a device hardware.

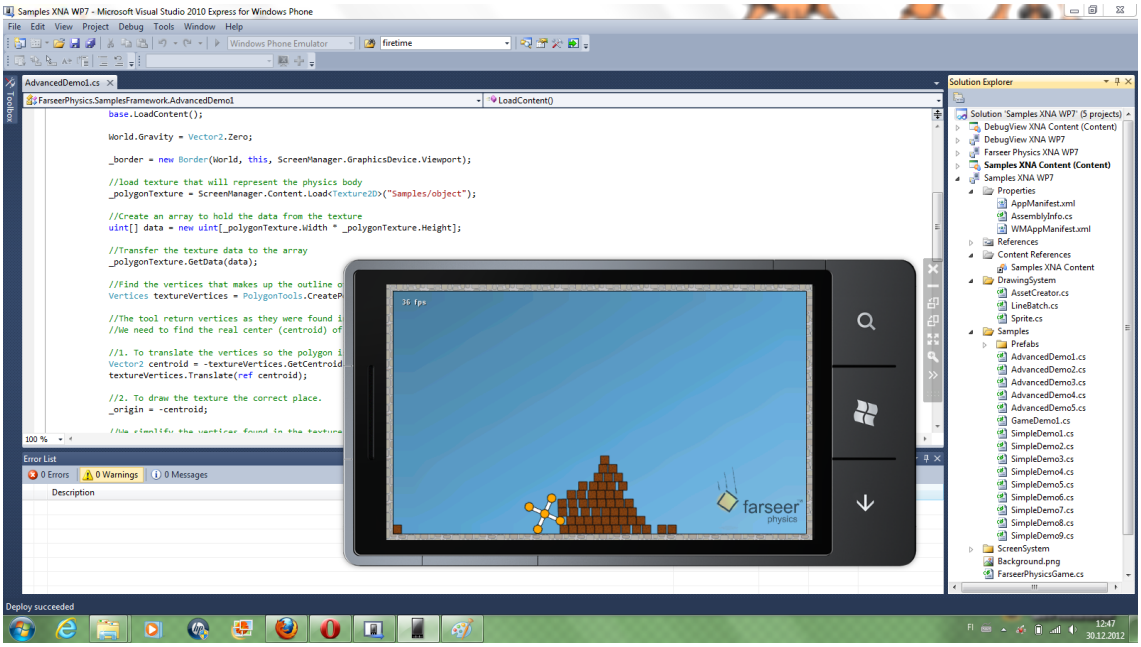



*FIGURE 1. The main tools: Visual Studio 2010 Express for Windows Phone IDE main view and Windows Phone Emulator launched on top of it.*

An appication is packed as a single *.Xap* package, which is actually just a zip package of all application content. Apart from the operating system, all the tools are free. The actual hardware device is entirely optional in the beginning of the development process as the emulator included in the SDK is highly suitable for most test cases including audio, a location based testing and an accelerometer sensor simulation.

Among other things, the SDK installation includes Visual Studio 2010 Express for Windows Phone, Microsoft XNA Game Studio, Silverlight SDK and the Windows Phone Emulator. Visual Studio (Figure 1) is the main tool with all the standard IDE features like a debugging support, an emulator and an HW deployment of software.

### 2.3 Silverlight and XNA

There are two frameworks that can be used in making a game. The event driven Silverlight is better suitable for ordinary applications with buttons, lists and pages. The XNA – a 3D ready real-time framework that comes with the Game Loop design – is better suitable for graphics heavy real-time running content like games. XNA Framework is designed to be the foundation on which XNA Game Studio applications, components, and controls are built (MSDN 2013b, date of retrieval 20.3.2013).

Previously, a developer had to choose one of the two frameworks whichever was better suited for the project at hand. Fortunately, Windows Phone 7.5 update called *Mango* broke the separation


*FIGURE 2. An example of classic Silverlight application UI (left), XNA UI (middle) and combined Silverlight/XNA UI (right).*

and now it is possible to mix the two frameworks. For example, an XNA framework "window" can be embedded in an Silverlight base project allowing a 2D/3D graphics screen to run in a 2D Silverlight application, and drawing Silverlight controls on top or on the side of the XNA graphics. The differences between the three different models are shown in Figure 2.

### 2.4 Game Loop

The Game Loop, illustrated in Figure 3, is an important design structure offered by XNA framework when developing an application on Windows Phone. It consists of three parts: Initialization, Update and Draw. The titles describe their purpose accurately. When creating an XNA game project in Visual Studio, the basic layout for a game loop structure is set, and the developer can start filling the methods with the code and engine logic instantly.

Initialization is run once for an instanced object. In this step a *LoadContent()* method is run, including a content resource loading like fonts, textures, sounds or models needed for the object. Also, some global values and object references can be set to be used by *Update()* and *Draw()* methods.

Update is the section where most of the object work logic is done. For a player character object in a game application, *Update()* method could call input handling methods, update movement based

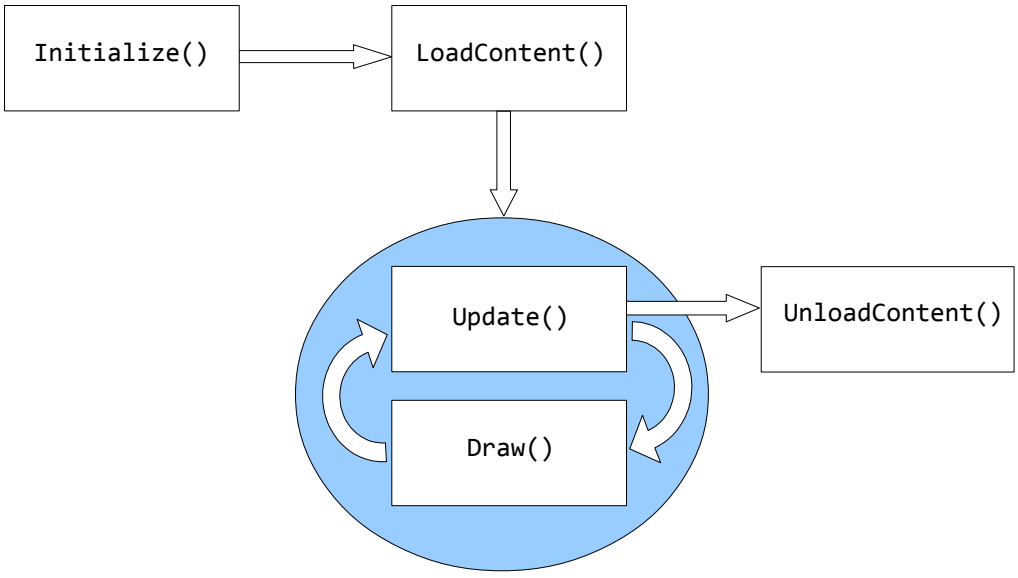



*FIGURE 3. Game Loop visualization*

on elapsed time, check for collisions with other objects, play sounds, update score and so on. This is the engine part of the XNA application.

Draw is the section where an object is drawn, whether it is a background image, a 3D object or a text string. The Windows Phone 7 screen update interval is 30 frames per second, meaning *Update()* and *Draw()* methods are called 30 times in a second by default. For a content heavy or complex application, the 30 fps target may suffer but the default value is great for any game and at the same level as current generation game consoles usually have for games.

When an object is deleted, the *UnloadContent()* method is called which will free up the resources by unloading any content loaded by *LoadContent()*. All Game Loop methods are defined in XNA framework's *Game* class. Thus a game willing to use XNA has to first create a class derived from *Game* to access the Game Loop functionality.

### 2.5 3D game world

Drawing a game world in 3D is substantially different than drawing in 2D. In 2D you only need to care about collisions in a 2D space and the movement is simpler. For objects, just calculate the place in X and Y coordinates and draw. A 3D space makes things much more difficult as there is also the Z-axis.

In XNA framework, the 3D graphics pipeline uses a graphics device to load resources and render a 3D scene or view. In general, the 3D pipeline requires the following state for initialization:

- view and projection transforms to transform 3D vertices into a 2D space.
- A vertex buffer which contains the geometry to render.
- An effect that sets the render state necessary for drawing the geometry. (MSDN 2013c, date of retrieval 20.3.2013)

The viewing transformation orients the objects so that they are arranged in the way that you want them to be displayed. It can be thought as positioning a camera inside the imaginary world and pointing it in a certain direction. After setting up the camera view, the 3D view needs to be projected from three dimensions into two as the phone only has a two-dimensional screen. There are two basic kinds of projections: orthographic and perspective. A perspective transformation makes objects farther away from a camera viewpoint appear smaller than those that are closer to the camera. This is the way your eye works, so it often gives a more realistic result than an

orthographic transformation. An orthographic projection simply drops the vertices' Z coordinates. This means that in contrast to perspective projection, parallel line segments retain their relative lengths so that an object that is farther away from the camera has the same size as a similar object closer to the camera. (Stephens 2011, date of retrieval 1.1.2013.) For 3D games, a perspective projection is the usual way to go, though orthographic projection could be used on purpose for example in special effects on the environment or objects.

In XNA there is no pure camera class, but there are properties that camera class should handle. View matrix basically includes information where the camera is positioned in the 3D world and which direction it is looking at. Projection matrix attributes include information about the rendered view: a field of view defines the angular field of vision, an aspect ratio defines the viewport width and height and is usually set to the dimensions of the device screen to avoid distorted or bent 3D object shapes and near and far plane distance values define how close and how far objects relative to the camera are rendered into the view. An example of a projection matrix definition in code with attributes in the same order is the following:

```
Projection = Matrix.CreatePerspectiveFieldOfView(MathHelper.ToRadians(50),
                graphicsDevice.Viewport.AspectRatio, 1, 10000);
```

Vertex buffer contains the data about the 3D objects that are about to be drawn to the graphics device. Vertex buffer can be accessed directly, but a normal 3D model loading can be done through the Content Manager of XNA framework. When loading a 3D model by calling the *ContentManager.Load<Model>* method, the content processor for the model is chosen automatically and the vertex buffer related activity is managed without any further actions by the developer. Basically, the whole thing happens with a few lines of code by defining a Model object and loading it through Content Manager:

```
Model = Game.Content.Load<Model>(@"Models\" + modelName);
```

BasicEffect class contains atrributes for rendering the objects. Class members include options for colours, lighting and texture among others. Lighting allows objects to have shadows and a visible form. For example, drawing a ball shaped 3D object looks like a flat circle without lighting. The quickest way to add lighting to a 3D "world" is calling the *BasicEffect.EnableDefaultLighting()*

method provided by the XNA framework at the drawing phase. No further tweaking is necessary, but if a developer wants to add a better lighting, there are several light sources available that can be added simultaneously to the 3D world: Ambient, Diffuse, Specular and Emissive. The specifics of these light types are out of the scope of this thesis but suffice it to say that the existing options are more than enough to fulfill the common needs for lighting a scenery or object. An example of Draw method drawing a 3D model in the MazeBall game:

```
foreach (ModelMesh mesh in Model.Meshes)
     {
         foreach (BasicEffect effect in mesh.Effects)
         {
             // Set the effect for drawing the component
             effect.EnableDefaultLighting();
             effect.PreferPerPixelLighting = preferPerPixelLighting;

             // Apply camera settings
             effect.Projection = Camera.Projection;
             effect.View = Camera.View;

             // Apply necessary transformations
             effect.World = FinalWorldTransforms;
         }

         // Draw the mesh by the effect that set
         mesh.Draw();
     }
```

### 2.6 Graphics and audio

XNA supports four audio formats: Wav, mp3, AAC and WMA. Due to size constraints, uncompressed WAV files work best for short sound effects, and compressed audio formats should be used for music. Loading sounds is straightforward. All audio is loaded the same way by calling the XNA framework Content Manager's *Load()* method and a built-in content processor handles automatically the loading of each file format. At least mp3 files are converted to a wma format by Visual Studio for a better size compression when packaging software as an XAP package to be installed on the device.

Audio can be played via *SoundEffect*, *MediaPlayer or MediaElement* classes depending on the purpose and source audio type. When dealing with audio, care must be taken not to violate platform audio policies. Music audio cannot be played freely in every situation and certain audio and volume controlling options need to be provided for the user if the application is targeted for

the Windows Marketplace publishing.

For graphics, BMP, JPG, PNG and GIF file formats are supported. Loading image resources happens the same way as audio through ContentManager and an appropriate content processor, using a *Texture2D* class. Textures attached to 3D object models work in a bit different way. 3D models in Windows Phone 7 have the Effect property that can be bound with the loaded texture in a few lines of code in the *Draw* method of the object. This also allows a quick and easy change of texture for a model if needed.

### 2.7 Controls

For possible controlling options for games, Windows Phone 7 devices have a touch screen and an accelerometer sensor. Controller scheme can be one specific or a mixture of any available input methods, for example touches on the screen and tilt of the device at the same time.

For a touch input, multitouch and a list of gestures are supported. Multitouch in Windows Phone 7 supports up to 4 simultaneous touch points (MSDN 2013d, date of retrieval 9.3.2013). 4 touch points may be a bit excessive for one player, but it enables the maximum of a 4-player real-time multiplayer game if controls are implemented for one touch point per player and anything in between. An example of a real-time 4-player game could be a reaction game. In such a game the device would be placed on a table, players evenly around it, and each player would have their own corner of the screen for themselves and the goal would be to tap or hold finger on the screen as shown in the example in the middle of the screen. The game would count the success rate of each player for example by counting the time the player is correctly following the touch orders of the game.

Gestures are special touch actions that can be enabled and disabled to provide the needed functionality. Windows Phone 7 supports 8 different gestures: tap, double tap, hold, horizontal drag, vertical drag, free drag, pinch and flick. The first six gestures are self explanatory. Pinch is a multitouch gesture where two fingers are placed on the screen and then either converged or diverged. Flick is a quick touch of a screen with a swipe movement, like a very short and fast drag.

Accelerometer can be used to detect a device tilt over X-, Y- and Z-axis and also the velocity of

the device, for example shaking the device so that the 3-angle tilt position is not altered. Visual Studio 2010 Express for Windows Phone IDE has a good visual accelerometer tool which can be used to reproduce the tilt and shake movement on different device positions like portrait or landscape flat modes, as shown in Figure 4.

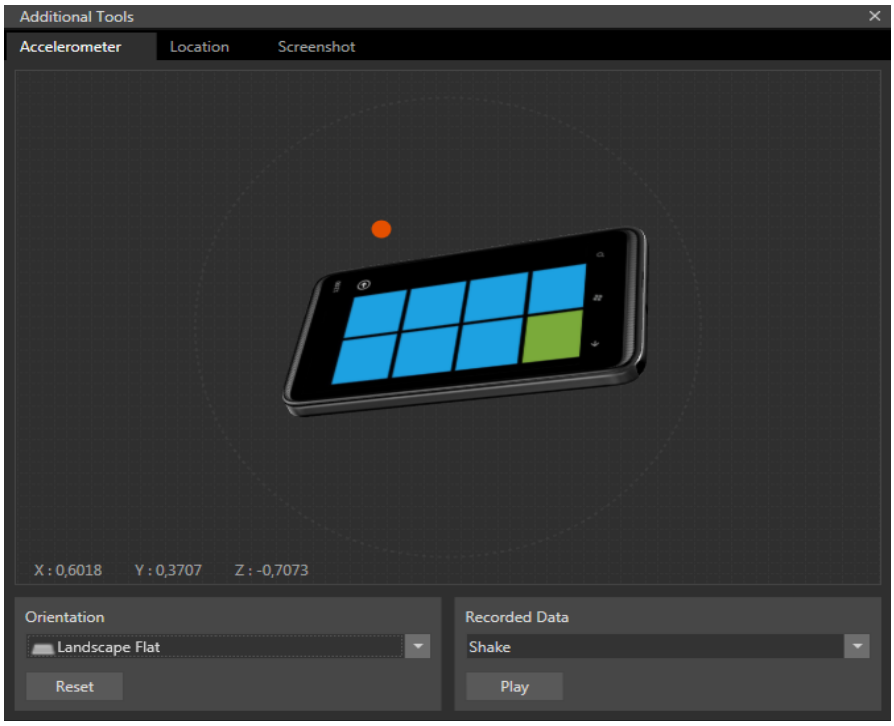



*FIGURE 4. Accelerometer emulator in Visual Studio 2010 Express for Windows Phone IDE*

# 3 OPEN SOURCE

## 3.1 History of open source software

The open source movement took shape in early 1980s university community through efforts of Richard Stallman to develop a free version of UNIX at the Massachusetts Insitute of Technology. During the ensuing years two branches of open source movement had formed: Stallman's Free Software Foundation (FSF) and the Berkeley movement, work done at the University of California. (Overly 2003, 3.) The Free Software Foundation promotes software freedom and resists proprietarty software, values that can be seen written on their General Public License (GPL) terms of use which state that all derivative works have to use the same license and offer the source code for free. FSF refers to its software as "free" in contrast to "open", as stated by the Berkeley movement and later on by Open Source Initiative (ibid., 3). The Berkeley movement's popular Berkeley Software Distribution license (BSD) is less restrictive than GPL as it also allows other licenses for derivative works.

Eric Raymond, a pro-business hacker focusing on the economic value and cost savings that Open Source Software represented, released his famous article "The Cathedral and the Bazaar" in 1997. The article reports Raymond's experiments with Free Software (the bazaar model) and reflects on the difference between it and the methodologies adopted by the industry (the cathedral model). The profit-driven, market-oriented uses of Free Software in mind, the Free Software Foundation did not represent freedom or liberty to Raymond in its mandatory idea of sharing everything for free. For him, the essentially interesting component of Free Software was not its enhancement of human liberty, but the innovation in software production that it represented. Soon after the release of the article, Raymond came up with the term Open Source and co-founded the Open Source Initiative. Prior to 1998, *Free Software* referred to either The Free Software Foundation or one of many other ideologies, research projects or processes having different names: sourceware, freeware, shareware, public domain software, and so on. *Open Source* sought to encompass them all. (Kelty 2008, 99 - 100, 107 - 109.) The Open Source Definition (Open Source Initiative 2013, date of retrieval 12.5.2013), originally penned by Raymond and Bruce Perens for the Open Source Initiative, is found in its entirety in Appendix 1. The Open Source Initiative has succeeded in spreading the Open Source term into common use of licensed software as seen on the present day terminology, articles and news.

Key events for open source movement's rise to mainstream was the release of Netscape Navigator web browser code for the public in 1998 amidst the wide adoption of key Free Software tools such as Linux for servers, Apache Web server for Web pages and the the Perl and Python scripting languages. (Kelty 2008, 107 – 108.) Open source has been since adopted by the businesses with many benefits in mind like low cost, rapid return of investment, increased reliability and decreased development time (Overly 2003, Introduction). Open source licensing is closely related to the corporate use of open source as, on the contrary to educational community and hobbyists, the corporate businesses use open source for monetary and/or other benefits. The commercial significance of open source has increased rapidly in recent years in both businesses and in public sector. For example, according to a research committed by Gartner in 2008, around 85% of businesses and communities were using open source solutions and the rest 15% were planning on that within the next year. (Oksanen & Välimäki 2010, 5.)

Open source has two sides to it. On the one hand it is a special way to develop software without the boundaries of organizations, and on the other hand it is a phenomenon stressed with challenging legal and business viewpoints. (Helander & Mäntymäki 2006, 1.) The former fulfills the original ideals of free software. The latter is more involved in contemporary topics like intellectual property rights and software patents and the term open source also covers this side of the coin.

Microsoft is a fine example of how a corporate business can use open source for their benefit. Windows / Windows Phone developer portal Dev Center has lots of code examples for different use cases that individual developers can use, modify and redistribute freely. These code examples are distributed under Microsoft's own Microsoft Permissive License, or Microsoft Public License (M-PL) as it is now known, that grants the user quite extensive rights to use the code and is very similar to other licenses of its kind, like Apache License. By doing this, Microsoft aims to make the Windows Phone platform more alluring by getting the developers quickly on track without any fees. Microsoft benefits from shorter software development times and increased SW offering and developers benefit as learning things is faster and easier compared to closed source where there are for example less insctructional code available and the ones that are, are not allowed to be modified, reused or redistributed.

### 3.2 Licensing

Licenses are a major, inseparable part of open source. Licensing happens for various reasons. For once, it protects a piece of software so that it is used only for purposes defined by the original author. For example, one of the oldest and most common open source licenses, General Public License gives permission to use the code for non-commercial purposes and also gives the user the right to modify and redistribute the code as seen applicable as long as the rules of the license are followed. Prohibiting commercial use prevents entities from making money on freely available code and the original free-for-all policy is strengthened.

The common thing between different open source licenses is that pretty much all of them grant the user the rights to use, modify and share the software within the limits of the license. These limitations differ between licenses.

The Open Source Initiative (OSI), a non-profit corporation formed to educate about, and advocate, for the benefits of open source, maintains both the Open Source Definition and the list

*TABLE 1. Common open source licenses (Open Source Initiative 2012, date of retrieval 7.12.2012).*

| License | Description | Example |
|---|---|---|
| Apache 2.0 | Approved in 2004 replacing old v1.1, this permissive license grants the right to use Apache 2.0 licensed software in commercial products without need to distribute source code as long as Apache License is included in The distribution | Apache Tomcat |
| BSD | "New" & "Simplified" versions of the BSD ( Berkeley Software Distribution) license available. This permissive license is quite similar to Apache License, granting users the right to do pretty much whatever they want with the code. | BSD |
| GPL (v2/v3) | GNU General Public License, version 3 released in 2007. The most widely used open source software license is so called "copyleft" license, meaning The derivative works of GPL licensed software must retain the license. Hence, distributing software under GPL license require that the source code must be available. | |
| LGPL | Lesser General Public License. More permissive than GPL, LGPL allows the linking of LGPL licensed components with non-GPL software, thus allowing distributor to not release the source code for the proprietary part of software. | |
| MPL 2.0 | Mozilla Public License, v2.0 released in 2012. Previous v1.1 license was uncompatible with GPL, v2.0 fixes that. | Firefox |
| Creative Commons | Available since 2002, Creative Commons helps licensing works freely for certain uses, on certain conditions or dedicate works to the public domain. CC currently comes in 6 different license types with varied level of restrictions. | CC licensed Music / graphics |

of Open Source Initiative-approved licenses. Among over 60 approved and valid licenses OSI website lists 9 most popular and widely used ones. Some of these licenses and their very brief descriptions are collected in Table 1.

The licenses are partly compatible with each other, meaning some licenses require that all derived works need to be distributed under the same license, others are more open and license term conflicts are avoidable. David Wheeler has published on his website (2007, date of retrieval 13.1.2013) a great chart on popular licenses compatibility with each other. This chart is presented with updated info in Figure 5.

In the chart, the shaded boxes are the names of different licenses (see Table 2). An arrow from box A to box B means that you *can* combine software with these licenses; the combined result effectively has the license of B, possibly with additions from A. For example, Apache 2.0- and GPLv2+-licensed software can be combined in GPLv3 licensed software. (ibid., date of retrieval 13.1.2013.)

On the left there are the “permissive” licenses, which permit the software to become proprietary (i.e., not FLOSS, Free/Libre Open Source Software). The “weakly protective” licenses in the middle compromise between permissive and strongly protective licenses. These prevent the software component from becoming proprietary, yet permit it to be part of a larger proprietary program. On the right there are the “strongly protective” licenses, which prevent the software from becoming proprietary. This includes the most popular FLOSS license, the GNU General Public

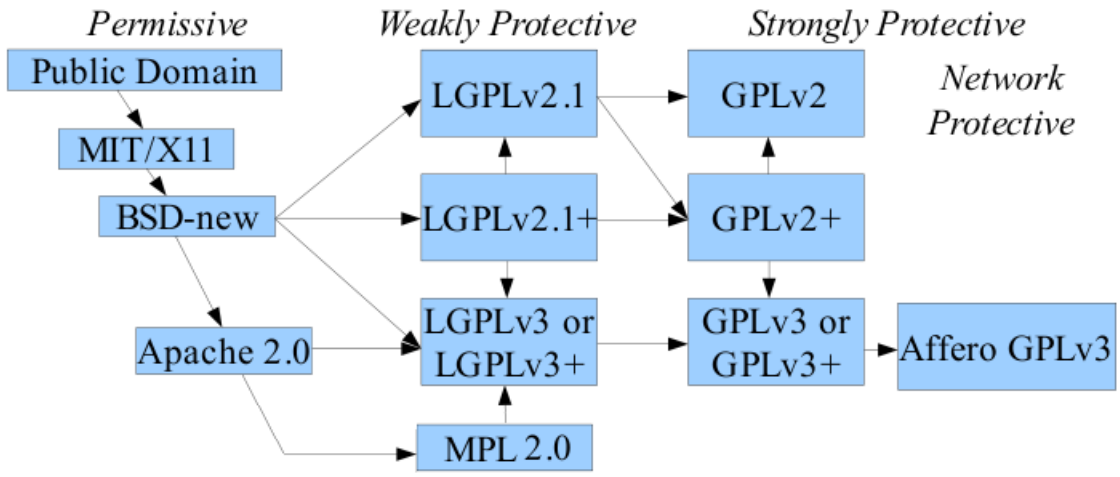



*FIGURE 5. License compatilibity chart.*

License. (Wheeler 2007, date of retrieval 13.1.2013.)

Becoming proprietary means that the open source code is utilized in software owned by a certain party and the party does not need to distribute the source code of their product for public. They can sell their software product, supporting services etc. without a problem. In the purest form, if someone wanted to, he/she could take an open source project with an appropriate permissive license, package it in their own liking and sell it with minimal to no changes done. The same software would still be available for free with source codes and all, so it would be bad business, but it could be legally done. Permissive open source licenses allow open source code to be used in proprietary, closed source products, but the resulting software is not open source.

In addition to open source software, more traditional resources like music, literary works or images, are available for similar kind of licensing. *Creative Commons* is a popular copyright licensing system used for these kinds of works. The CC licensing system has currently 6 different license types to choose from with varied restrictions and additionally a special *CC0* license which is equivalent to a public domain (Creative Commons 2013, date of retrieval 7.3.2013). All the licenses require users to give credit to the originator of the work (attribution), but vary in allowance of monetary benefit (commercial/noncommercial options) and further derivations of the licensed work (noderivatives, sharealike options).

### 3.3 Legal issues

Licensing issues related to open source require, in addition to technological and software engineering skills, a set of legal, business and sociological capabilities from companies and individual developers to manage the people and processes related to open source software development. (Helander & Mäntymäki 2006, 1.) A good real-life example of a conflict, a lack or a neglection of these skills may cause, happened in Finland in the summer of 2006. A person found out by looking through a product firmware of a few network equipment manufacturers' devices that they include the GPL licenced code. The catch was that the source code the license usage required was not publicly available. The person asked the companies to provide the source code for the products that violated the GPL license. Shortly after consulting their attorneys one of the companies published the source code. News articles following this event can be found on several websites (Mäki 2006; Sektori 2006, date of retrieval 11.1.2013).

An example of a copyright violation regarding other type of resources is a well publicised plagiarism controversy that happened in 2007 between a Finnish digital music composer and an American music producer over a song by artist Nelly Furtado. Lawsuits were filed which claimed that the American producer had plagiated parts of the Finnish musician's song for his own product without any credit for or consent of the original author. A collective article with valid references can be found in Wikipedia (2013, date of retrieval 7.3.2013).

In this thesis I have wanted to emphasize the use of open source, the usefulness of it and other benefits. This chapter covered the topic of open source in general. To naturally bridge the open source and game development and its sections, it is appropriate to show how many open source tools were used during the game project development and the whole thesis creation process. The list completed with the tool description and commercial equivalent is collected in Table 2.

One of the most basic differences between open source and commercial software is the cost. The license prices of commercial software listed in Table 2 vary from 30 USD to over 3,000 USD depending on software and its features. The most expensive license, 3,675 USD in April 2013 (Autodesk 2013, date of retrieval 1.4.2013), was for Autodesk Maya, a 3D animation suite rival to open source equivalent Blender.

*TABLE 2. Open source tools used during game development and thesis creation and their commercial equivalents.*

| OSS component | Description | Commercial equivalent |
|---|---|---|
| Blender | 3D content creation suite, providing modeling, texturing etc. functionality in one package | Autodesk Maya |
| BEPUphysics | 3D physics library with collision detection and collision detection etc. features | Matali Physics. Basic version is free of charge, Pro version has payable license. |
| Firefox | Popular web browser | Internet Explorer |
| WinMerge | Comparing and merging tool for files or folders | Beyond Compare |
| OpenOffice | Office software suite for word processing, spreadsheets, presentations etc. | Microsoft Office |
| 7-Zip | File archiver utility | WinZip |

# 4 GAME DESIGN

Designing a good game requires thinking through many different aspects, such as a gameplay, a graphical style, characters and an environment or level design. Even if a game was brilliant in every other way, even one heavily flawed feature or development area can ruin the whole game. For example, a far too difficult or a too easy game can lead to frustration if the challenge is not at the suitable level. One of the most frustrating things the player can experience is to replay some short part of the game over and over again due to the unevenly balanced level of difficulty. Even worse is the situation where the game has been far too easy up until the point where the player gets stuck. To avoid these kinds of shortcomings, general design steps should be followed when creating a game.

## 4.1 Areas of game design

Game design covers widely all aspects of a game from overall features such as a replayability or a target player demographic to more obvious topics like a sound design and an art style. If designing a commercial game, or a game for specific target demographics, there are things like a target device platform, budget, marketing, and so on to consider. This thesis, however, is about an independent, open source game development where game design boils down to the core gameplay fundamentals and features. Some of the most important design fundamentals according to O'Luanaigh (2006) are collected in Table 3. In addition to game design aspects listed in Table 3, there are four key gameplay areas that need to be focused on: Camera systems, Control systems, Character design and Environment & Level design (O'Luanaigh 2006, Introduction). Besides gameplay specific focus areas, audio is also an important part of nearly any game.

### 4.1.1 Camera

Camera work can make or break your game. It represents the players' eyes and is something that is used every second they play the game. There are different problems with different camera systems but most problems are usually related to a few commonly occurring things. A badly positioning camera getting stuck behind objects, bouncing around or pointing to a wrong place can ruin games. A camera too tightly attached to a player character can result in a jerky camera as even a small bump will instantly alter the camera. A camera too loosely attached can drift too

*TABLE 3. Key areas of game design. (O'Luanaigh 2006, 18 - 24.)*

| Design aspect | Description |
|---|---|
| Interaction | Focus on interaction. It makes games interesting. Without being able to interact with the environment and objects, player will get bored quickly. |
| Variety | Maintaining variety is fundamentally important. Old games forced the player to play the same few minutes of gameplay over and over again. Today, players are different and variety is needed to keep modern players interested and challenged. |
| Player Frustration | Avoid frustration. Players should always feel they could complete the game in one go if they didn't make mistakes. Having places where players cannot fail but die the first time they encounter them is wrong. |
| Replayability | Add replaybility by awarding players with repeated play, for example giving bronze, silver or gold award for completing a level in a certain way |
| Exploration | Encourage exploration. Players need a reason to want to continue with a game. You might tease the player by showing them a glimpse of great weapon or some really jaw-dropping gameplay and use that as a lure to drive them forward. |
| Collectibles | There is an innate desire within all of us for collecting things. Use collecting and completing techniques to enhance your game. For example, create a number of bonus item and hide them around the game world and let player see how many they have acquired. |
| Difficulty | Use difficulty levels wisely. There are different types of gamers. Some are not very good at games but still love to play, some are very skilled and demand challenge. Consider using difficulty levels so that both types of players can enjoy themselves. |
| Fun | Games must be fun. Get people to play the game and ask what bores and annoys them? Do they enjoy it and is the game still fun after playing for a while? |

far away or suffer from too much lag when it rotates, thus reducing visibility until it catches up. (O'Luanaigh 2006, 126 - 127.) These are problems mainly for 3D camera views. Still, a loosely attached camera or a bad design of 2D view can affect a 2D game to a similar degree.

### 4.1.2 Controls

Equally important to having a brilliant game camera is having a superb control system. The control system is how your player character responds to the player when a button is pressed or an analogue stick is moved on the controller. A game with the right control system feels instinctively right as the player character moves just as expected. (ibid., 147, 153.) Control system possibilities vary per platform. PCs have a keyboard and a mouse, consoles have a controller with variable amount of buttons and analogue sticks, and tablets have a touch screen. Target device is important when designing a control system. A great example of a failed control system alteration happened in *Uncharted* action adventure game series on the Playstation 3 platform. The first two games in the series had a similar feel to aiming a gun and moving the

target reticle across the screen. The third game in the series, while otherwise maintaining the series characteristics, saw a change in the aiming mechanic. It immediately felt sluggish and just plain bad for many players. The aiming issue raised so much controversy that the developer applied a patch to give players an option to toggle on an aiming style of the previous games (Naughty Dog 2011, date of retrieval 9.6.2013).

### 4.1.3 Characters

Designing a game around an unusual main character was extremely common in the early days of computer games when player characters varied from frogs to eggs, and even babies. The majority of main characters nowadays are human adults. One reason for this is the increased emphasis on immersion, with cinematic cameras and the growing popularity of first-person perspective. Creating a unique game character is a major challenge, but there is a real opportunity to set a game apart from generic design. (O'Luanaigh 2006, 173 - 175). Character design is important for any game where player controls a creature or object with human qualities. Certain type of games, like a simple puzzle or card game, can skip character design completely. However, even a simple poker game can be enhanced with memorable characters, as demostrated by *Telltale Games* with their *Poker Night* games where the players are represented by characters from various game series (Telltale Games 2013, date of retrieval 9.4.2013).

### 4.1.4 Levels

Environment and level design progresses step by step, first creating a level design blueprint. Gameplay mechanics need to be designed and decided first, so that a level is created to support the gameplay, not the other way round. An important thing is to keep a constant set of rules, such as using a light to help indicating the places the player needs to go, and never breaking them to make players realize what the light means every time they see it. Another key thing is to make the player's goal clear so that they always have an idea what to do or where to go next. After defining the blueprint, a level needs to be created. While the gameplay fundamentals of the camera and control systems provide instant feedback to the player, the level design will define the experience over a longer term. The process of level creation includes creating rough sketches first, then the first rough geometry to be able to test play the level. Then, fine-tuning the level based on the test play will help to create the final balance of pacing, unique moments, locations, and so on. (O'Luanaigh 2006, 204 – 216).

### 4.1.5 Audio

In real life, a person is constantly bombarded with thousands of pieces of information every second through senses, such as smell, taste or touch. Sound and music in a game are important as there are only a small number of ways to give feedback to the player. Without sound, there is only a mainly visual feedback available for the player. Music can be used to give a dynamic feedback to the player. For example, changing music dynamically from a normal background music to a faster action scene music can give the player info that an enemy has spotted the player character, even if that enemy is located off-screen. Sound effects can be used for a similar effect as music. Simple things like having different footsteps on different surfaces will immeadiately give the player a feedback of what they are standing or running on. Sound effects and music can make a real difference by letting the player to realize and react to what is going on more quickly, and they can deliver tension, fear and excitement. (O'Luanaigh 2006, 299 - 300, 316.)

## 4.2 Design and creation flow

Game development can be illustrated with traditional software development models, such as the waterfall or spiral model, but a simple flowchart by game designer David Midgley sums the process effectively. Midgley's *Game Project Flowchart* is shown in Figure 6. The chart shows well

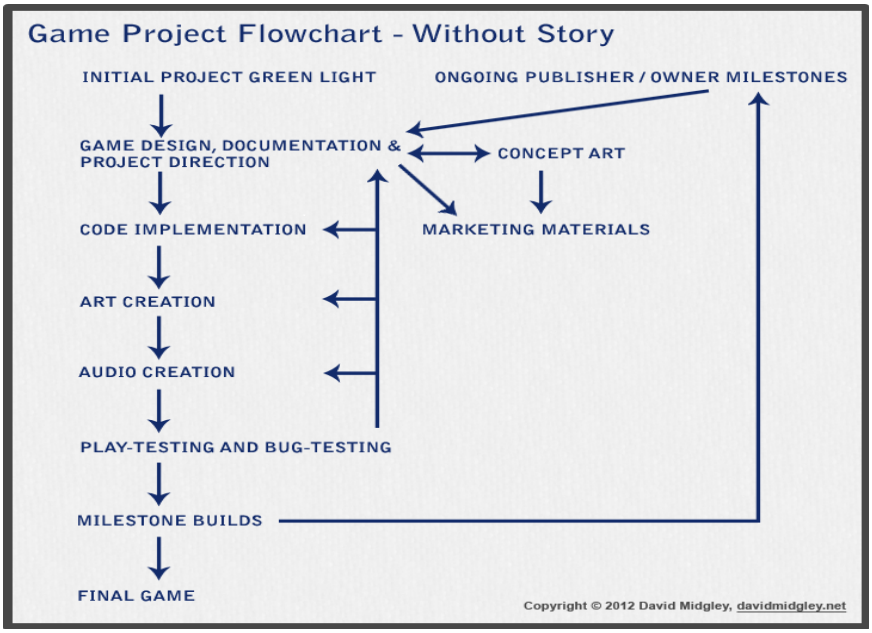



*FIGURE 6. Game Project Flowchart (Midgley 2012, date of retrieval 9.4.2013)*

the iterative and cyclic, repeating nature of game development. A continous play-testing affects coding, graphics, audio and even the initial design as the feedback from testing is assessed and the necessary actions are taken.

# 5 DEVELOPED MAZABALL GAME

## 5.1 Requirements

The game development project was done as a one man work, so formal game design documents and the actual development was done in a bit of a chaotic way. Constantly learning new things about the tools, available open source components and the target platform caused some re-thinking of certain game ideas and functionality. Being a designer, a developer, a tester, and an asset creator at the same time meant that only a few roles at a time could be focused on. This sometimes caused a lack of focus on other areas. For example, during the most hectic coding phase some previously thought ideas were dropped on the fly as there appeared another way of accomplishing something similar, or the idea was found to be impossible to realize.

The game project was started by listing the minimum requirements the game needed to have. The list was short, including only the core functionality and features: a main menu that loads up when the game is launched, the actual game view of a 3D maze level the ball travels through, a visible running counter of playtime and acceleration sensor controls for moving the ball inside the maze and some form of audio. Everything else would be optional extra.

From device point of view, the contol scheme was clear from the beginning: an accelerometer sensor. There is really no easier, more intuitive way for a player to control a moving ball on a rotating "floor" than the accelerometer that can be used to track a player movement by a device rotation. That was the core of the control scheme. Other things, like vibration, could be added to enhance the gaming and UI experience if needed. From the beginning there were a few ideas for a vibration feedback during the game but it was decided that they would be added as extra features during the final polishing stage if there were time.

The UI itself, or the game screen was hard to get right the first time. I wanted to do a 3D view of a maze with a viewport turning a bit when a phone was turned. This was the basic "view" I wanted the player to see. In addition to that different things were thought to be shown on the screen and how they would look like. Too much info – a score, a possible timer, a "life" count and so on – would possibly draw attention from the game screen too much by cluttering the sides of the screen. Too little info could look plain and be too non-informative. A player had to have some sort of indicator about the level progress, a timer or such to have an inspiration to go and finish the

level or try to do a record breaking speedrun! Initially, I settled for a splash screen for the level number when starting a new level and a running timer for the finishing time and leaderboards.

The graphics requirements were low: possibly a wooden or some kind of tile maze "floor", wooden or metal walls and a shiny silver, chrome or marble ball. A background image for the main menu was mandatory but no other needs were initially thought.

For audio, an in-game music while playing was initially the only true need. It was not necessarily a mandatory feature but such a game experience richening thing that it was going to be seriously looked after. Audio in general was considered as an optional feature that could be enhanced as much as possible after the main game functionality had been first done.

### 5.2 Resulting game

To begin with, I only had had few short courses on the Windows Phone development at work so I pretty much started from scratch what comes to game development. First thing was to dig into the Microsoft Windows Developer portal, the official developer community, and see what it had to offer. There a perfect starting point was found in a few different game development courses targeted at newcomers: a step-by-step guided course of building up a side scrolling shoot 'em up game that taught the usage of graphics, sound and most importantly, the Game Loop. After checking out some possibilities with a combined UI model, which would allow for example a pause menu with a Silverlight textbox or button controls over a full screen XNA game window, a decision was made to stick to a full XNA framework for the simplicity of the Game Loop.

Secondly, an even greater help was found. As a part of Advanced topics in the self-learning code examples kit from Windows Phone Developer Community there was a hands-on lab exercise for a 3D game creation. The game was also a maze game similar to what I had in mind. I studied the lab and realised there was still a lot of work to do even if I had a good starting point in the lab exercise. After finding the Maze game example, I decided to take that as a base project and remove a more complex content from it and also try to utilize as many open source tools and resources as possible to replace the original example project content. Besides programming the application itself, a lot of work was needed with the resource aspect of the project: graphics, sounds, and the entire 3D game world. The options were limited; either to create everything from scratch or to utilize available resources. In this development project, a bit of both were done.

The final game I ended up with has the following main features: one 3D maze level without textures, running counter for time on the upper left corner of the gameplay screen, in-game and menu music, some sound effects, accelerometer controls and a highscore list. The game is launched via *Play* selection from the main menu, which loads the main game level and shows a zooming "3...2...1...Go!" counter on the screen before giving the control to the player. In the game, the player controls a ball through the maze by tilting the device so that the ball rolls to a desired direction while avoiding holes on the floor. If the ball drops through a hole, its position resets to the beginning of the maze. When the player reaches the end of the maze, a congratulations note is shown and if the playing time for beating the game is within top ten in the highscore table, the player is asked to enter their initials to the table. The game also has a calibration screen for the accelerometer which can be accessed by double tapping the game screen, and a pause screen that can be accessed by pressing a left arrow *Back* hardware button during gameplay.

**5.3 UI**

The game uses a *ScreenManager* class that handles all screen class objects in the game. The Screen Manager was taken as it is from the game lab example, but the existing screens were removed, added and modified. The final version of the game has 8 screens in total: main menu, high score, options, about, loading, gameplay, calibration and pause screen (see class diagram in Appendix 2). As an example, the main menu and game screen are shown in Figure 7.

The resulting game has a main menu with Play, Highscore, Options and About selections. All applicable screens can be backed to the previous view with the left arrow "Back" hardware key. An on-screen "back" button is also present on most screens.

*Play* launches the game by loading the game resources and starting the game. Highscore options shows the top 10 fastest players and their times. Options and About screens are completely new ones. Options screen includes the audio settings that are saved to a file and loaded upon launching the game. The audio settings are: toggle game music and sound effects on/off, and an option for overriding system audio. By default audio policy, an application cannot interrupt external audio that is already playing when that application is launched. Overriding system audio means that if the user is, for example, listening to music with a Music Player application and launches the MazeBall game, the music already playing can only be paused, stopped or resumed


*FIGURE 7. Main Menu and Gameplay screens*

by the game application if the override option is enabled. The application, which is not following this audio policy, will not pass the Microsoft certification to Windows Marketplace. The following is a code example of this policy following an audio playback:

```
public static void DisableMusic()
{
    // Disable music option activated in Options menu
    if ((MediaPlayer.GameHasControl || OptionsScreen.OverrideEnabled) &&
         !OptionsScreen.GameMusicEnabled)
    {
        MediaPlayer.Stop();
    }
}
```

About screen has a vertically scrollable text describing the game background and listing the open source and related components used in the game and their license info. Scrolling was made by reading and handling vertical drag gesture events and moving the place of the single long text string accordingly.

One notable feature is disabling the user mode idle detection in the main gameplay screen. By default, the Windows Phone 7 power saving settings are configured so that if the user does not touch the screen or press hardware keys for one minute, the device screen is shut down and the device is locked automatically. This is a problem for applications, usually games, using acceleration sensors as an input method. Playing for a minute just by tilting the phone will cause a device lock and a screen shutdown. By disabling the user mode idle detection, these battery saving features are not activated and the user can play the game normally as long as needed. The following is a code example of a user mode idle detection disabling:

```
// Countdown ends, game begins. Disable UserIdleDetection to prevent screen lock after
```

```
// 60s of play.
            PhoneApplicationService PAService = PhoneApplicationService.Current;
            PAService.UserIdleDetectionMode = IdleDetectionMode.Disabled;
```

### 5.4 3D modeling

Blender is an open source 3D content creation suite that can be used to design and create 3D models and animations. As stated in Chapter 2, the XNA framework only supports *.fbx* format for 3D models. Blender can also do this and it was a clear winner as the modeling tool. In the MazeBall development project Blender was used to create the 3D maze playfield, and the ball the player moves through the playfield. Both models were exported as an *.fbx* format and imported as a resource to the game project in Visual Studio.

The main 3D model of the game is the mazel level itself. It was the single most time consuming thing to do in the project, not for the complexity of the model but due the complexity of the tool itself. Having no previous experience with any 3D modeling tool, hours were spent by learning first the very basics of the tool itself and only after that moving on to the actual modeling of the maze. An image of the maze model creation in Blender is shown in Figure 8.

The ball object was simple to create. Only a ball shaped object was needed, nothing more. A texture could have been added to the ball object in Blender, but it was decided to leave texturing

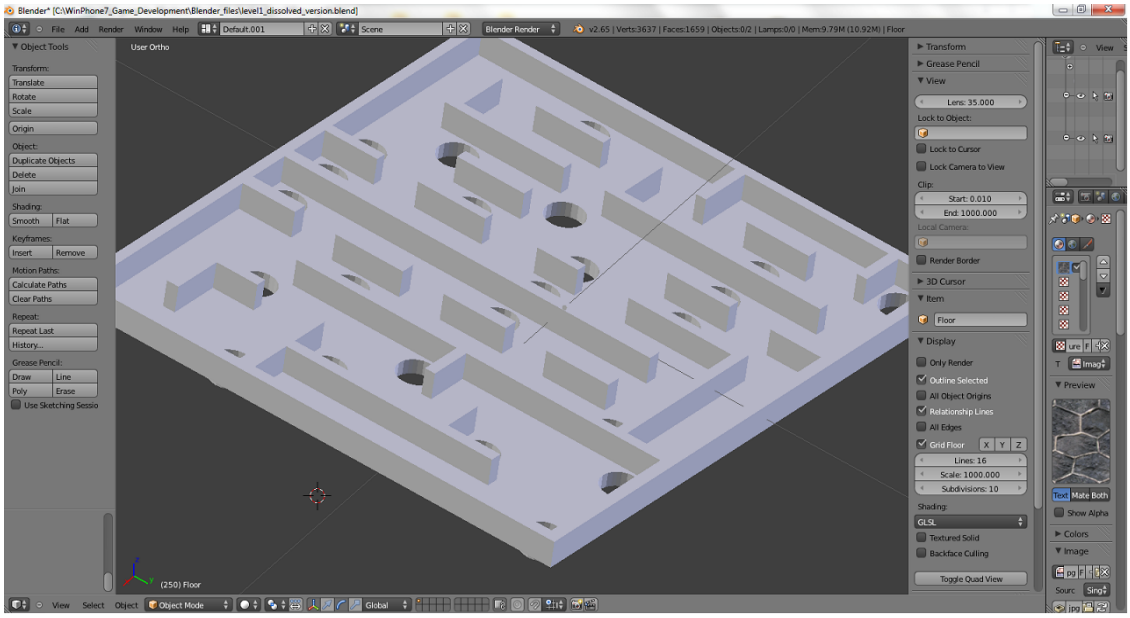



*FIGURE 8. Blender user interface – desiging Maze level.*

To the Visual Studio side to be able to re-texture, or re-skin, the ball for the purpose of using different materials on the ball and thus different gravitational values making the ball feel lighter or heavier. The original *.blend* files containing the maze and ball models are included with the source code distribution for reference, as the final, exported models included in the Visual Studio project, which Windows Phone supports, are in the proprietary Autodesk *fbx* format that cannot be imported back to Blender.

### 5.5 Music and audio

The sound was a hard thing to tackle. I had an idea of playing some level music in the background and to play a few possible sound effects on top of that – bouncing to a wall, dropping to a hole, finishing the stage, and so on. Not being able to compose any kind of music myself, finding freely licensed music to be played as a stage music was the only viable option. In the beginning the pure minimum for the sound portion of the game was to have at least some sound effects, with or without level music.

Useful sound effects were found in the base project resources, but there was need for an option to use other sounds, too. From the *freesound.org* website I found a few usable sound effects that fitted well for the game. In the end I settled using only a short menu click sound effect available through the Creative Commons CC0 license, meaning the effect was a public domain without any restrictions.

Finding suitable, royalty free music was not as difficult as was initially doubted. After some searching there appeared to be several websites dedicated to royalty free music where artists share their works through the Creative Commons license. The initial idea for music was to use some kind of an ambient electronic track in the background during the menu navigation and some faster, upbeat instrumental techno or rock track in the actual game. During the audio section development phase there were two placeholder filling-in songs that fit the description but as they were traditionally copyrighted, they could not be used in the final version.

After a lengthy browsing and listening of samples, the two chosen game music tracks were not either of the initially chosen genre but they fitted in the game just as well. These chiptune songs, mimicking the sound of the old Commodore 64 computer, are hosted by the Free Music Archive website *freemusicarchive.org*, dedicated to a varying degree of free music with hundreds of titles

available for listening and downloading. Both songs, *Konamized* by *Rushjet1 and Moves* by *Sycamore Drive* are licensed by the *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 United States (CC BY-NC-SA 3.0 US) license* allowing the noncommercial use. If I was going to charge any money for the game in Windows Marketplace or just use these songs without giving credit to the original authors, the usage of these songs would be a violation of the CC license.

In the About menu section of the game I have listed all the music and sound effects used with author attribution and info where to find the files. This fulfills the attribution clause of the Creative Commons license on music tracks but it is a good practise on any content not necessarily requiring it.

### 5.6 Graphics

The Web search of open source licensed graphics provided acceptable results. From the OpenClipArt website, which contais CC0 public domain licenced graphics, I found a tiny thumbnail image for Games menu selection in the device. I also looked at several other sites[1] for maze textures. Some considerations for the maze floor and wall textures found on these sites are collected in Figure 9.

The main menu background image was rendered in Blender with wall and ball textures missing. I had a big trouble with the texture binding for maze walls and in the end I decided to drop the textures from the maze object altogether, keeping it barren and plain but clear. Maze textures are the first thing to do properly in further development.

In the About menu section of the game I have listed all the graphics used with the author

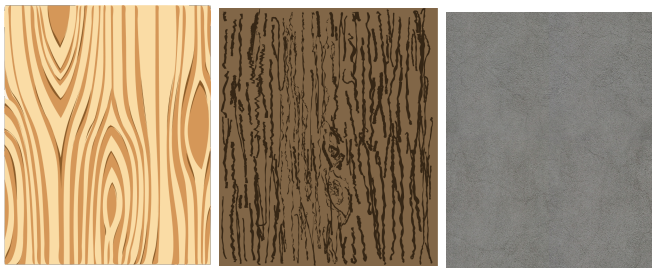


*FIGURE 9. Samples of licensed or public domain graphics.*

1 www.openclipart.org, www.pdclipart.org, www.deviantart.com

attribution and info where to find the files. Even though only a few graphics samples were used in the game, some more are included in the project resources as possible maze textures and ball skins for a future work.

### 5.7 Open Source physics

BEPUphysics is a freely available open source 3D physics library distributed on the Apache 2.0 License. The BEPU homepage[1] is the source for a source code, samples, a documentation, community discussions etc. The site was invaluable for me when I started to study how BEPUphysics could be used in my MazeBall game.

One of the main features of BEPUphysics is the collision detection. Trying to do the math or calculations needed for detection if certain objects collide in a 3D world is indeed a whole lot more complicated than checking if two object overlap on a 2D space. I did not look too much into the collision detection features available in the SDK libraries. Instead, I went straight for the BEPUphysics features which seemed easy to understand and simple to use.

The single most significant change to the base project was the integration of BEPUphysics library. The Microsoft developer community game lab project had complex, hard to understand code for a 3D model loading, collision detection and model interaction. With the introduction of BEPUphysics, it was possible to get rid of all the code I did not quite fully understand and instead, let the open source physics library deal with all that.

The integration of BEPUphysics was relatively easy and straightforward. Setting up the basic physics simulation only needed the *Space* class object handling the actual physical space, set gravity for the space and suitable object class instances to put into that space. For this game, only the ball object has physics affecting it. The maze model is set as a static, kinetic model meaning that the gravity does not affect it and it does not move, but dynamic objects, like the ball, will collide with it. The following is a code example of setting up the physics simulation space and the object representing the ball:

```
bepuphysics_space = new Space();
bepuphysics_space.ForceUpdater.Gravity = new Vector3(0, -981f, 0);
```

1 http://bepuphysics.codeplex.com

```
.
playBall = new Sphere(level1StartPosition, 25.1f, 50);
.
bepuphysics_space.Add(playBall);
```

BEPUphysics is such a feature rich library that adjusting different variables and options and also utilizing some features of the library would no doubt offer better, more realistic physics simulation results but given the time and focus of this thesis, it was convenient to stick with the basic working solution. Despite the helpful website forum community, the matter that documentation for the library is scarce, primarily pushing new users to study few code examples, affected the decision, too. "Easy to learn, hard to master" would be the phrase describing the library.

One bug that is present in the final version of the game seems to be related to the physics engine. Rarely, when the ball drops through a hole on the floor, some calculation relating to the position or collision of the ball goes wrong and instead of valid X, Y and Z values of position vector, the engine gives a result NaN, Not A Number, and the game freezes.

### 5.8 Debugging and testing

Debugging and testing were mainly made in an emulator by two methods: a functional testing and breakpoints. Audio and music testing was made by a functional testing. By checking all cases where music should play, pause or resume and a different sound should play, it was verified that all audio cases were working correctly. Setting breakpoints in an IDE debugger and investigating different class member values were useful, especially in finding the incorrectly set values for certain variables and invalid update logic for 3D objects. A few issues were tracked by drawing debugging traces on the screen, for example the ball location was tracked real-time by drawing its X-, Y- and Z-coordinates on the screen while playing. This method revealed the "not a number" (NaN) bug still occurred in the application sometimes when the ball position calculation had suddenly failed. Other problems and errors were usually related to typos, an incorrect syntax and missing resource definitions that were revealed at build time.

A hardware device was not used for debugging at all, but only for testing after emulator results were good enough. The mandatory hardware testing was needed only for an audio policy testing where a music player was needed to be playing audio as the application was launched to see if

the override audio option was working correctly.

### 5.9 Open source in Windows Phone game development

The research question of this master's thesis was "How can open source be utilized in a Windows Phone game development?". Upon studying the theory and developing the MazeBall game for Windows Phone 7, three major ways of utilizing open source occurred. These three ways, or benefits are the licensed resources, open source software and open source tools. In Figure 10 these three beneficial ways of using open source are presented.

One way to utilize open source are available licensed resources. This means that there are various websites offering different kind of audio and graphical content for varying degree of freedom. For example, *openmusicarchive.org* offers hundreds of Creative Commons licenced songs in many genres spanning from ambient techno to metal music.

Another way to use open source is the actual open source software itself that allows a developer

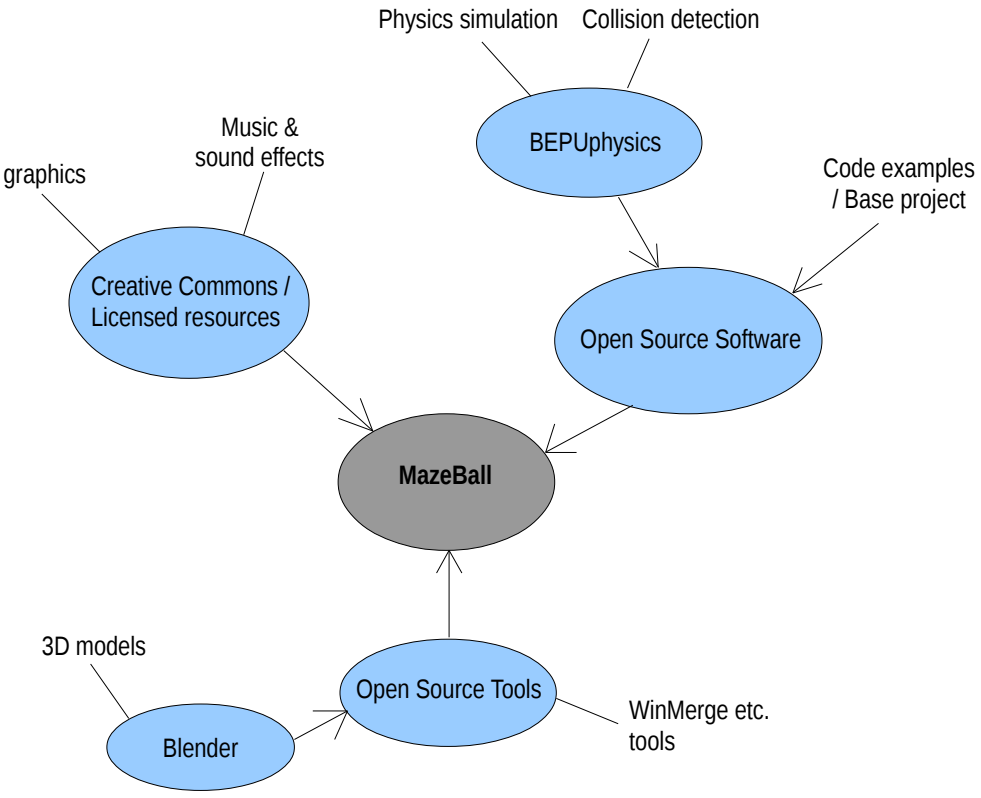



*FIGURE 10. Utilization of open source components in MazeBall game*

to study, use and derive existing code. The Microsoft developer community game lab project with complete source code proved to be a major part of the MazeBall game framework. It provided a working base I could start to extend and replace with the needs of MazeBall project in mind. The second major help was the BEPUphysics dll library which simplified the collision detection and physics calculations greatly.

The third way to use open source were open source tools. The open source tools are free and can optionally be modified and enhanced to user needs. In the MazeBall game development the Blender 3D creation suite was a key tool in creating object model assets. These tools may offer equal or even better features and functions than commercial closed source counterparts as was shown by the Firefox web browser which had for example a tabbed browsing long before the industry leader Internet Explorer introduced the similar feature.

# 6 DISCUSSION

This master's thesis presented a Windows Phone 7 platform game development project from an idea to an alpha stage finished product. An open source topic was also covered at the general level, and some legal issues that may appear when using it were presented. I wanted to find out how open source can be utilized in a Windows Phone game development. The research results were that open source can be utilized in three ways: licensed resources, open source software and open source tools. In addition to these three ways, also an open source community could be stated as the fourth way to utilize open source. Of course, the community support is available almost everywhere but the open source community seems to be more active and helpful in my opinion.

It was surprising to find such a variety of open source software and licensed resources available for game development, and to see there are open source based tools for most needs a student or developer can imagine from a 3D modeling to a full office suite. No commercial or paid licenses beyond the Windows operating system were used while making this thesis. In best case, the use of open source can be a mutual benefit for both the developer and the community maintaining the code the developer gains the benefit of using the code and the community in return gains for example bug fixes or port of the component to another platform.

There is also a downside to utilizing open source. As open source is mostly a community developed work made for free and for common good and goodwill, there may be some parts to it that are lacking compared to commercial software. Some challenges met during the game development included the lack of documentation or examples for certain software components which led to some wasted time trying to find answers from community forums. Other issue may be the user friendliness of software UI, or the lack of it. For example, I tried a few open source UML modelling tools while working on the game. After trying them with no success, the third application I tried, a commercial one, was clear enough to figure out and I succeeded in what I wanted to do quickly. Other challenges are related to learning all the new things. I had to study lots of new tools like 3D modelling and code comparison tools.

A maze game was chosen as a game type, and Windows Phone 7 as a target platform. For a different kind of game for a different platform, the ways of utilizing open source would have been the same. Open Source is not dependant on a platform or a game genre. Licensed graphics and

audio can be used anywhere, code examples can be ported from one programming language to another, and open source tools are usually available for multiple operating systems. The BEPUphysics library is XNA specific, but other open physics engines are available for other platforms.

Overall, the game development project was a demanding effort for a single developer. Despite the copious amount of help given by open source, there were new tools to learn, APIs to study, code to write and resources to find, browse through and utilize. Engineering studies and work experience on mobile device software were highly beneficial while doing the project.

What I believe I succeeded best in this project was that despite being new to many aspects of game design and coding, I managed to do a working prototype of the MazeBall game. Some requirements had to be dropped or modified during the process and game polishing had to be left for future because of limited time resources.

The most challenging thing was to learn the necessary skills to make the Blender 3D object textures work correctly. I had a limited success in doing the maze floor, for example, but I would have needed much more time to get around fixing texture issues for the maze walls. The concepts like UV maps and tiled textures would have had to be studied and learned and as that started to sidetrack the actual development work, I settled with the plain 3D model for the maze.

Further development possibilities for the Mazeball game are vast. The very first thing to do would be fixing, or finalizing, the maze level 3D model to contain the textures for walls and floor. After that new levels and music tracks could be added. Also, the ball physics could be enhanced and the accelerometer values adjusted so that the game would be more responsive. You could maybe even add an sensitivity option to vary the sluggishness of the ball. A single most imminent visible change would be making a better main menu background image as the current image serving as one is basically just a practise render of the Blender camera object viewport.

This thesis is a small development project example with some open source learning alongside. This thesis can be beneficial to some readers, possibly students planning on doing their first published software project. I found utilization of open source very interesting and useful and I can recommend it to anyone. Download the game. Check the source code. Study, modify, improve or build upon it. It is all free and it is all there for that purpose, distributed under the open source

license, naturally.

## REFERENCES


AdDuplex. 2013. Windows Phone Device Stats for March 2013. Date of retrieval 20.3.2013
http://blog.adduplex.com/2013/03/windows-phone-device-stats-for-march.html

Autodesk. 2013. Autodesk Maya purchase page. Date of retrieval 1.4.2013
http://www.autodesk.com/products/autodesk-maya/buy

Creative Commons. 2013. Creative Commons Licenses. Date of retrieval 7.3.2013
http://creativecommons.org/licenses

Gartner 2013. Press release on Gartner report "Market Share Analysis: Mobile Phones, Worldwide, 4Q12 and 2012". Date of retrieval 19.3.2013
http://www.gartner.com/newsroom/id/2335616

Gamasutra 2013. News article: "It's Official. XNA is dead". Date of retrieval 20.3.2013
http://gamasutra.com/view/news/185894/Its_official_XNA_is_dead.php#.UREwnmResyF

Helander, N. & Mäntymäki M. 2006. Empirical insights on open source software business. Tampere University of Technology (TUT) and University of Tampere (UTA).

Kantar. 2013. Kantar Worldpanel New Centre: "Windows sees strong European growth"Date of retrieval 19.3.2013
http://www.kantarworldpanel.com/global/News/Windows-sees-strong-European-growth

Kelty, C. 2008. Two Bits: the cultural significance of free software. Durham and London: Duke University Press.

Midgley, D. 2012. Game Development Flowchart. Date of retrieval 9.4.2013
http://www.davidmidgley.net/wp-content/uploads/2012/01/game_development_flowchart1.gif

MonoGame. 2013. About. Date of retrieval 20.3.2013
http://monogame.net/about

MSDN. 2013a. XNA Game Studio 4.0 Refresh. Date of retrieval 20.3.2013
http://msdn.microsoft.com/en-us/library/bb200104.aspx

MSDN. 2013b. XNA Framework Class Library. Date of retrieval 20.3.2013
http://msdn.microsoft.com/en-us/library/bb203940.aspx

MSDN. 2013c. 3D Pipeline Basics. Date of retrieval 20.3.2013
http://msdn.microsoft.com/en-us/library/bb194916.aspx

MSDN. 2013d. Working with Touch input. Date of retrieval 9.3.2013
http://msdn.microsoft.com/en-us/library/ff434208.aspx

Mäki, M. 2006. Telewell aloitti gpl-lisenssin alaisen lähdekoodin julkaisun. Date of retrieval 11.1.2013
http://www.digitoday.fi/data/2006/08/08/telewell-aloitti-gpl-lisenssin-alaisen-lahdekoodin-julkaisun/200610326/66

Naughty Dog. 2011. Uncharted 3 Patch 1.02 Notes. Date of retrieval 9.4.2013
http://www.naughtydog.com/site/post/uncharted_3_patch_102_notes

Neogames. 2011. Finnish Games Industry 2010-2011. Date of retrieval 19.3.2013
http://hermia-fi-bin.directo.fi/@Bin/e67ee5bd77e16e568b93f6ca796962c1/1363684588/application/pdf/928763/Finnish%20Games%20Industry%202010-2011.pdf

Oksanen, V. & Välimäki, M. 2010. Avoimen lähdekoodin oikeudelliset riskit. Helsinki: Työ- ja elinkeinoministeriö.

Open Source Initiative. 2012. Licenses. Date of retrieval 7.12.2012
http://opensource.org/licenses/category

Open Source Initiative. 2013. The Open Source Definition. Date of retrieval 12.5.2013
http://opensource.org/docs/osd

Overly, M. 2003. The open source Handbook. Pike & Fisher, Inc.

O'Luanaigh, P. 2006. Game Design Complete. Scottsdale, AZ: Paraglyph Press Inc.

Sektori. 2006. ADSL-modeemivalmistajien tuotteissa GPL-koodia. Date of retrieval 11.1.2013
http://sektori.com/uutinen/adslmodeemivalmistajien-tuotteissa/7181

Stephens, R. 2011. Into The Third Dimension. Date of retrieval 1.1.2013
http://msdn.microsoft.com/en-us/library/hh420951.aspx

Telltale Games. 2013. Poker Night 2. Date of retrieval 9.4.2013
http://www.telltalegames.com/store/pokernight2

Wheeler, D. 2007. The Free-Libre / Open Source Software (FLOSS) License Slide. Date of retrieval 13.1.2013
http://www.dwheeler.com/essays/floss-license-slide.html

Wikipedia. 2013. Timbaland plagiarism controversy. Date of retrieval 7.3.2013
http://en.wikipedia.org/wiki/Timbaland_plagiarism_controversy

APPENDIX 1 The Open Source Definition (Open Source Initiative 2013, date of retrieval 12.5.2013)

The Open Source Definition

Introduction

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

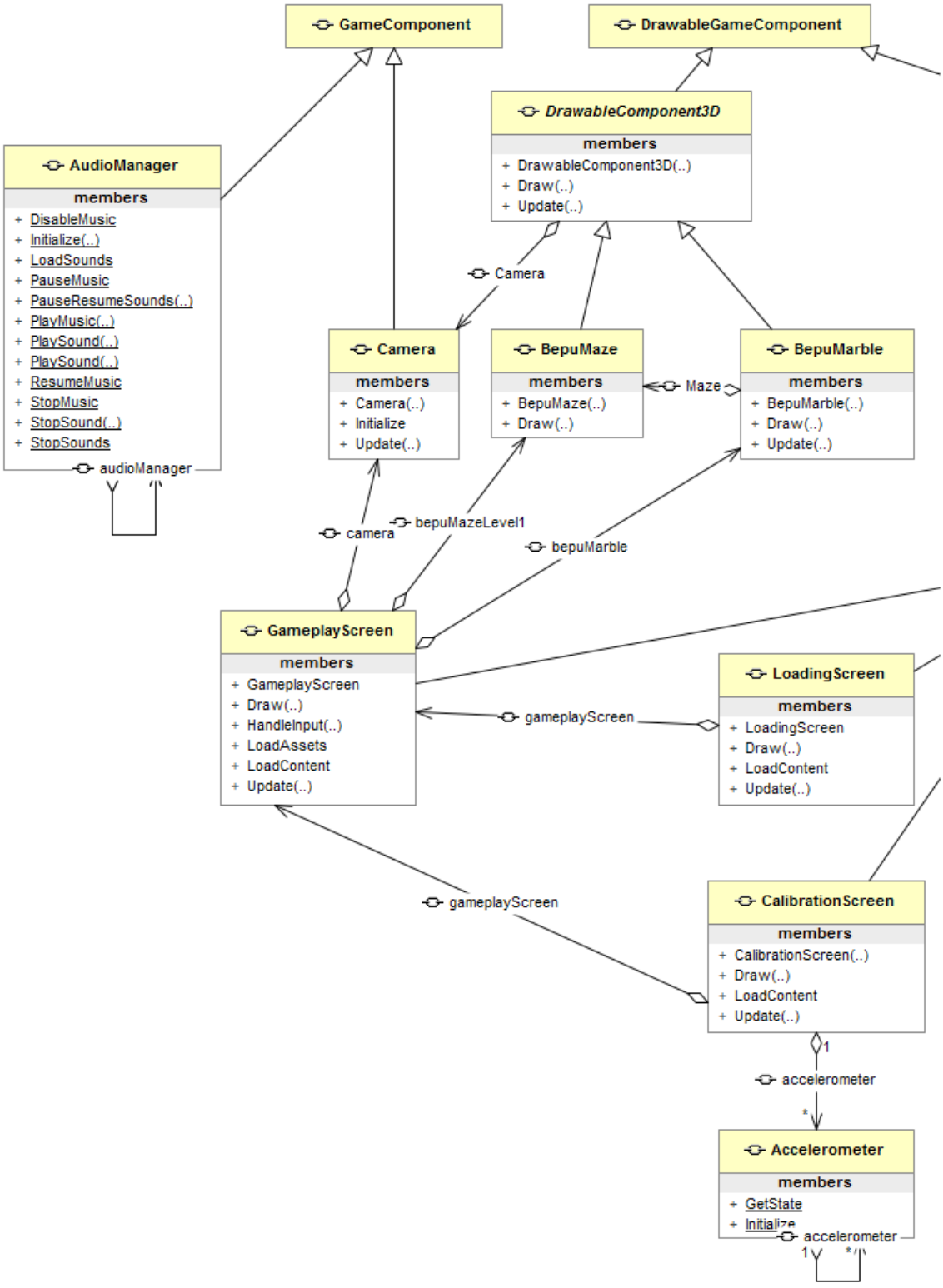
9. License Must Not Restrict Other Software

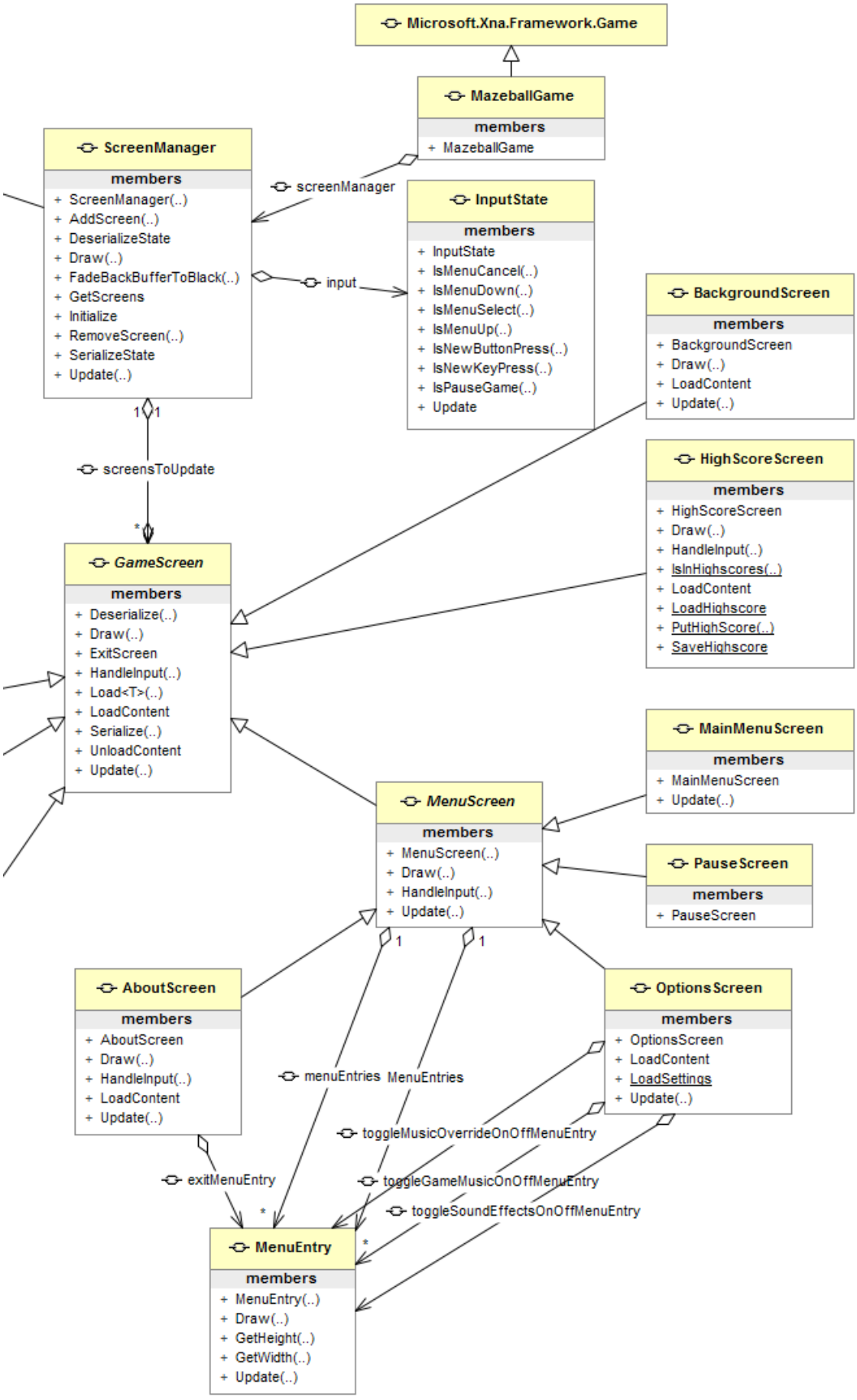
The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

10. License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface.

APPENDIX 2 Mazeball class diagram




Microsoft.Xna.Framework.Game
MazeballGame
members
+ MazeballGame
ScreenManager
members
+ ScreenManager(..)
+ AddScreen(..)
+ DeserializeState
+ Draw(..)
+ FadeBackBufferToBlack(..)
+ GetScreens
+ Initialize
+ RemoveScreen(..)
+ SerializeState
+ Update(..)
screenManager
InputState
members
+ InputState
+ IsMenuCancel(..)
+ IsMenuDown(..)
+ IsMenuSelect(..)
+ IsMenuUp(..)
+ IsNewButtonPress(..)
+ IsNewKeyPress(..)
+ IsPauseGame(..)
+ Update
input
BackgroundScreen
members
+ BackgroundScreen
+ Draw(..)
+ LoadContent
+ Update(..)
1
1
screensToUpdate
*
HighScoreScreen
members
+ HighScoreScreen
+ Draw(..)
+ HandleInput(..)
+ IsInHighscores(..)
+ LoadContent
+ LoadHighscore
+ PutHighScore(..)
+ SaveHighscore
GameScreen
members
+ Deserialize(..)
+ Draw(..)
+ ExitScreen
+ HandleInput(..)
+ Load<T>(..)
+ LoadContent
+ Serialize(..)
+ UnloadContent
+ Update(..)
MainMenuScreen
members
+ MainMenuScreen
+ Update(..)
MenuScreen
members
+ MenuScreen(..)
+ Draw(..)
+ HandleInput(..)
+ Update(..)
PauseScreen
members
+ PauseScreen
1
1
AboutScreen
members
+ AboutScreen
+ Draw(..)
+ HandleInput(..)
+ LoadContent
+ Update(..)
OptionsScreen
members
+ OptionsScreen
+ LoadContent
+ LoadSettings
+ Update(..)
menuEntries
MenuEntries
toggleMusicOverrideOnOffMenuEntry
exitMenuEntry
toggleGameMusicOnOffMenuEntry
toggleSoundEffectsOnOffMenuEntry
*
*
MenuEntry
members
+ MenuEntry(..)
+ Draw(..)
+ GetHeight(..)
+ GetWidth(..)
+ Update(..)